DIGITAL LOGIC DESIGN

THE FRUIT HUNTER

Presented By

Abeeha Zehra az07728

Batool Zehra bl07621

Muhammad Youshay my07103

Dhanani School of Science and Engineering

Habib University

Fall Semester

December, 2022

Abstract

“As part of our DLD project, we created a fruit hunter game based on the concepts of bubble shooter and space invader”

Table of Contents:

1. Introduction
   1.1. Project Introduction
2. User Interface
   2.1. Proposed diagram
   2.2. Implemented diagram
3. Input block
   3.1. Joystick
   3.2. Dual axis xy analog joystick
   3.3. Joystick as input
   3.4. Pin configuration of analog dual xy joystick
   3.5. Pin configuration of analog dual xy joystick and fpga on our implemented game
   3.6. Circuit layout and design
   3.7. Joystick implementations
   3.8. Button as input
   3.9. Simulations
   3.10. Code snippets
   3.11. Input block
4. control block
   4.1. description
   4.2. states
   4.3. state diagram
   4.4. state table
   4.5. state equations
   4.6. circuit diagram
5. output block
   5.1. output
   5.2. pixels
   5.3. block diagram
   5.4. schematic
6. Acknowledgements
7. Conclusion
8. References

SECTION 1-INTRODUCTION

1.1. Project introduction

We intend to develop a game based on the concept similar to space invaders, and bubble shooters. Our game will have a user-friendly interface. Our main character will be a hunter who would try to collect fruits present on the top of the screen using an arrow that he would carry. However, in order to obtain the fruits, he must overcome the obstacles that stand in the way of the fruits and the hunter. This game would have three lives, each life would be deducted when the arrow hits the obstacle rather than any fruit. When all three lives are used up, the game would end. Our game is based on the functionality of a mealy machine. This video game is executed by dividing it into modules such as input module, output module, and control unit. We used a joystick and button as our input peripheral, and we employed it to FPGA Basys 3 board to connect it to the computer. The output module is responsible for the output interface, which is our screen and the pixels generated on basis, the control unit will be responsible for managing the flow of the game, which include our game rationales, state diagrams, and FSM.

SECTION 2- USER FLOW DIAGRAM

2.1. Proposed Diagram

→Here is our proposed idea, as outlined in our project proposal.

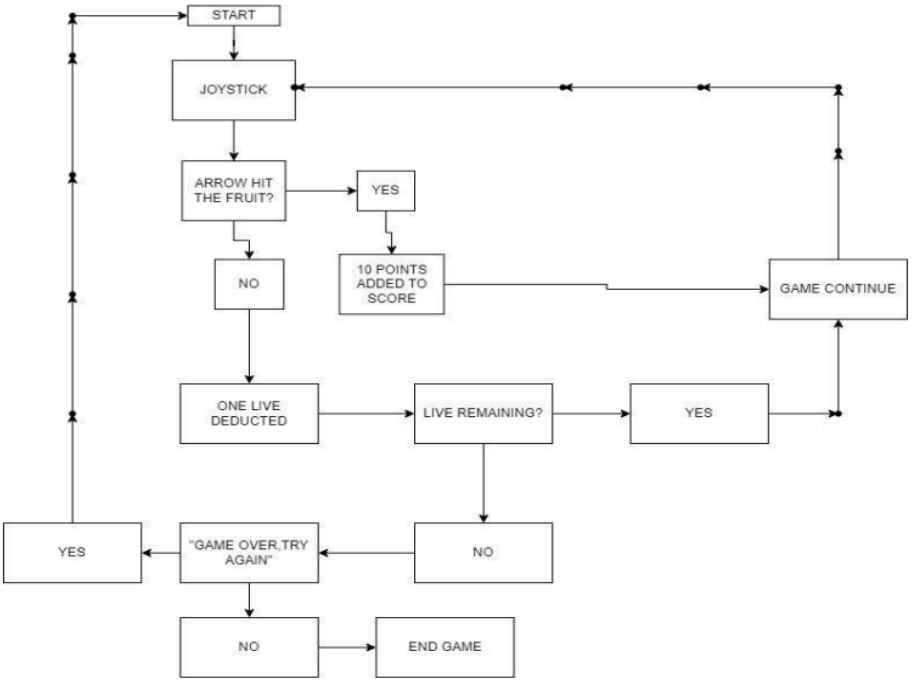

Figure 1: Proposed idea

2.2. Implemented Design:

→ Here is the user flow diagram for the implemented design.

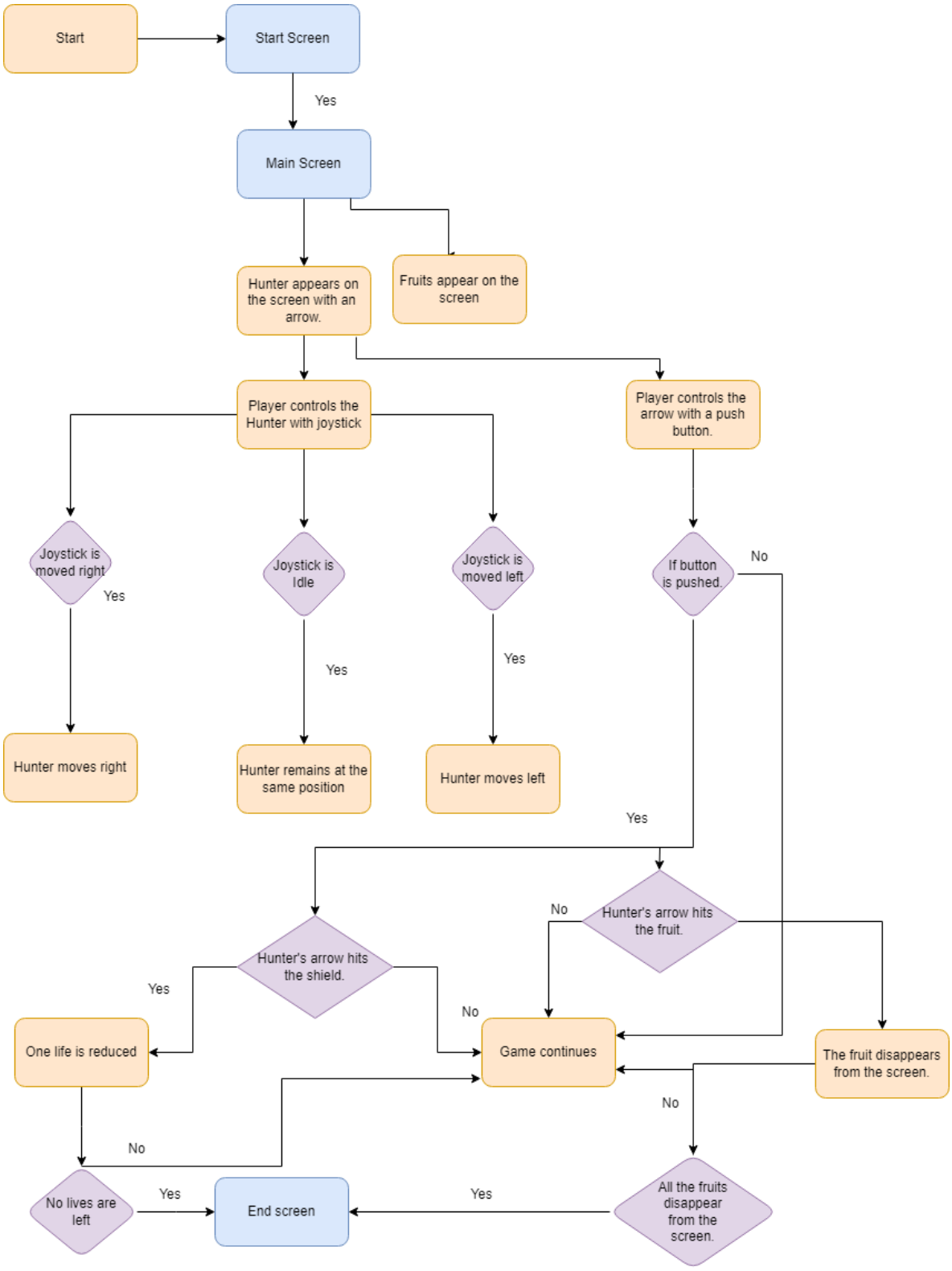

Figure 2: implemented design

SECTION 3: INPUT BLOCK

3.1. JOYSTICK:

We chose a joystick as an input peripheral because it is very prevalent in the videogames epoch to play games with a joystick. Interestingly, joystick has a very user-friendly and fast interface that makes it simple to navigate. Users can provide control in three dimensions with a joystick, and it provides a more enjoyable gaming experience than any other input. We used a dual-axis XY analogue joystick for our project.

3.2. DUAL-AXIS XY ANALOG JOYSTICK:

An analogue joystick is similar to a Dual-axis XY Analog Joystick module It has one push button and two variable resistors. Push buttons can produce output in either low state or high state digital form, while variable resistors produce output in the form of analogue voltage. One potentiometer controls the x-axis, and the other one controls the y-axis. At the centre, there is a push button that controls the z-axis dimension.

Figure 4: Dual axis xy analog Joystick

3.3. JOYSTICK AS INPUT:

The joystick is linked to PC via FPGA basys 3. The Analog to Digital signals (xADC) on the BASYS 3 FPGA Board was employed to convert the analogue voltage signals produced by the joystick into digital signals to be utilized in our game. The FPGA Basys 3 can only interpret voltages between 0 and 1 volts, but our joystick port can supply over 5 volts. We needed a narrow set of voltages to connect the FPGA and joystick, therefore we employed the voltage divider technique, which divides the voltage first and then delivers it to the FPGA

board. To divide the voltage, we need a resistance of approximately 15.6kohms. So, to achieve this we have used two resistors with resistances of 15kΩ and 680Ω that were placed in series.

3.4. PIN CONFIGURATION OF DUAL-AXIS XY ANALOG JOYSTICK:

1. Gnd: GND is the module's Ground Pin, which must be connected to the negative (ground) terminal of the 5v power supply.
2. +5v: Positive input of module's pin that must be connected to the positive (+) terminal of the 5v power su
3. VRx: This pin provides an analogue output voltage range of 0 to 5v in response to joystick knob/cap movement in the X-direction (axis)
4. Vry: This pin provides an analogue output voltage range of 0 to 5v based on the Y-direction movement of the joystick knob/cap (axis).
5. SW: SW is the switch's digital output. When it is normally open, the digital output from the SW pin is HIGH. When the button is pressed, it connects to GND, resulting in output LOW.

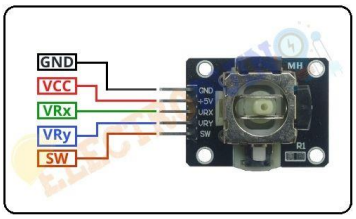

Figure 5: Pin configuration of Joystick

3.5. PIN CONFIGURATION OF JOYSTICK AND FPGA BASYSY 3 BOARD ON OUR IMPLEMENTED DESIGN:

1. Gnd: The Ground pin of the joystick is intended to be connected to the FPGA Board's JXAC5 pin pin of FPGA basys 3 Board
2. +5v: it will be connected to one end of the resistor (voltage divider circuit as mentioned above) and the other end will be connected to JXAC6 pin of FPGA BASYS 3 board
3. VRx : VRx pin of joystick is connected to JXAC1 pin of FPGA basys 3 Board
4. VRy : VRx pin of joystick is connected to JXAC2 pin of FPGA basys 3 Board

5. JXAC8 and JXAC7: Both pin is interconnected which is further connected to JXAC5 pin of FPGA basys 3 Board

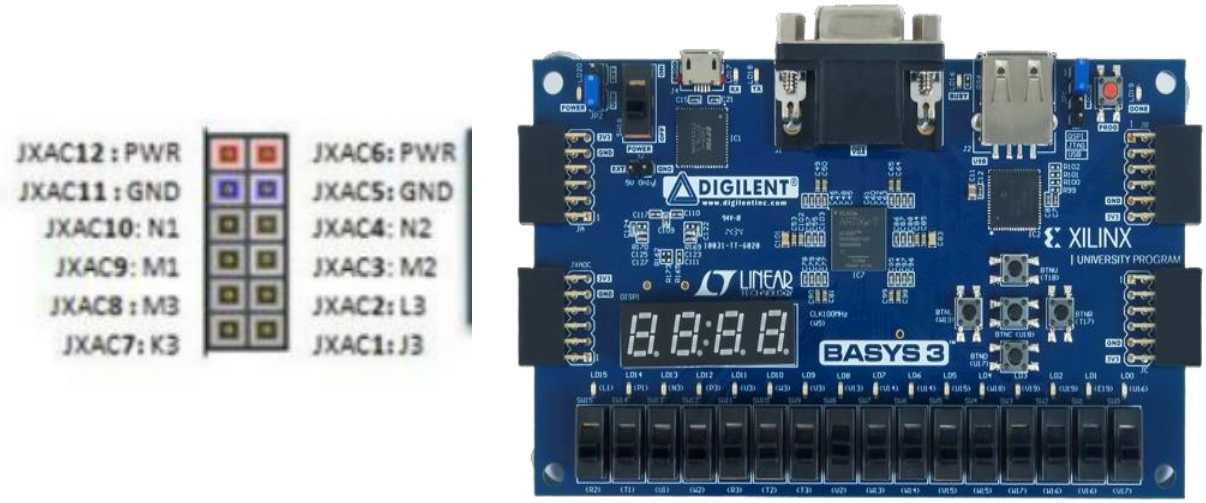

Figure 6: Pin Configuration

3.6. CIRCUIT DIAGRAM/LAYOUT

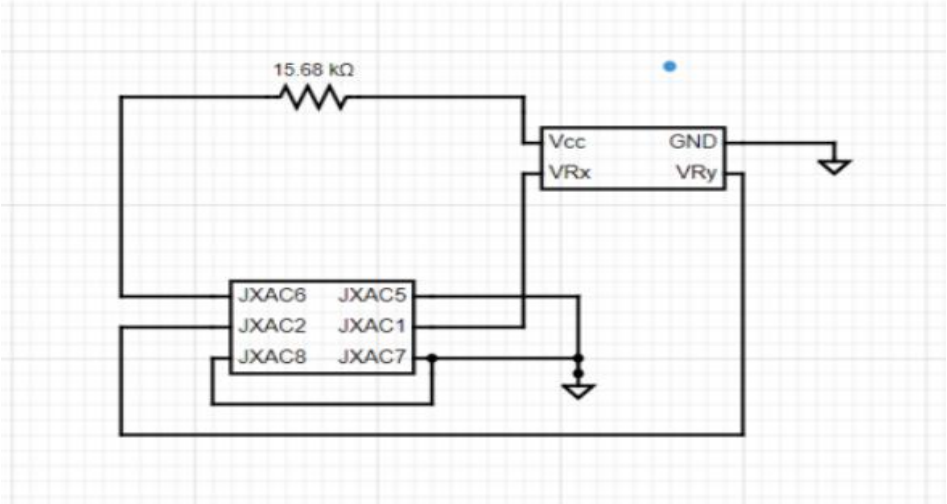

Figure 7: Circuit Design

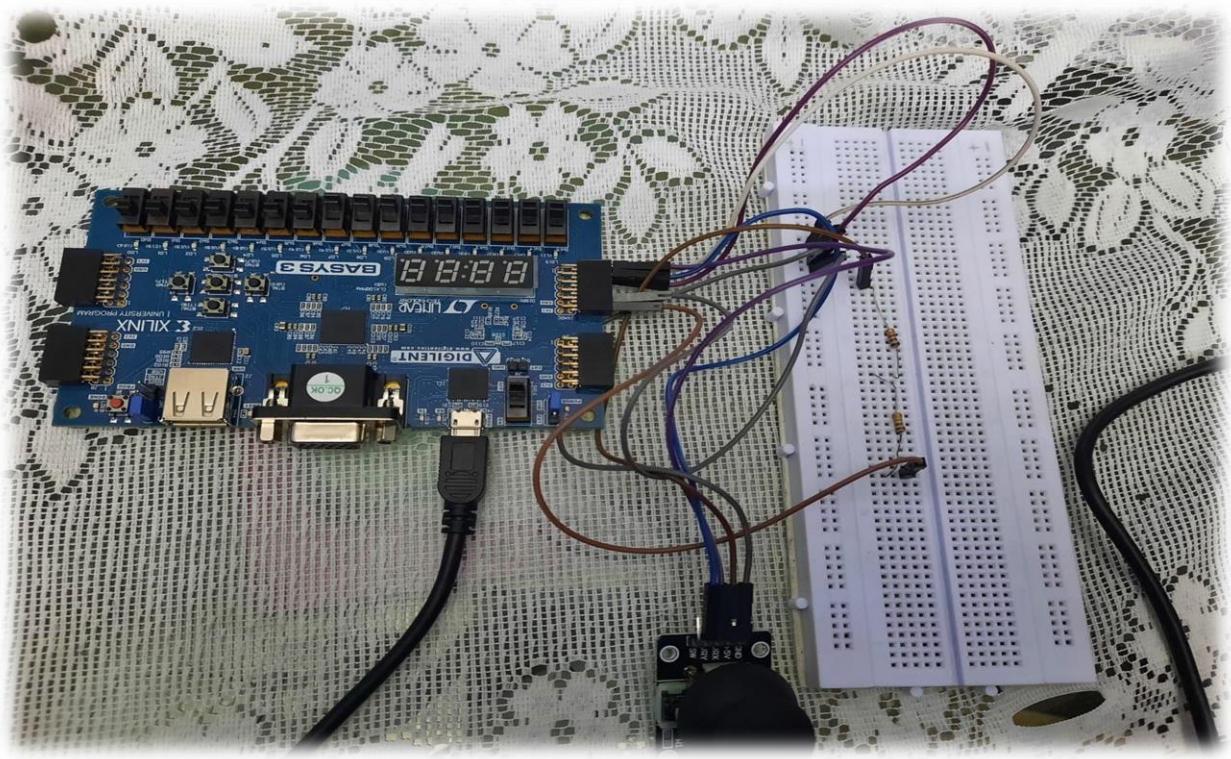

Figure 8: Implementation on Breadboard

3.7. JOYSTICK IMPLEMENTATION:

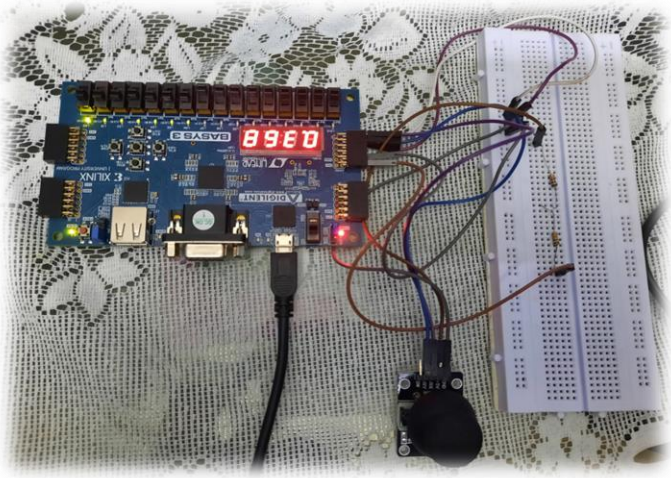

Figure 9: When joystick is idle

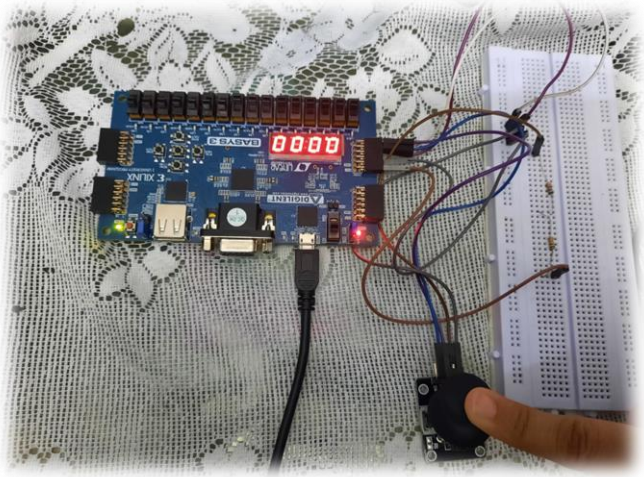

Figure 10: When joystick is moved in right direction

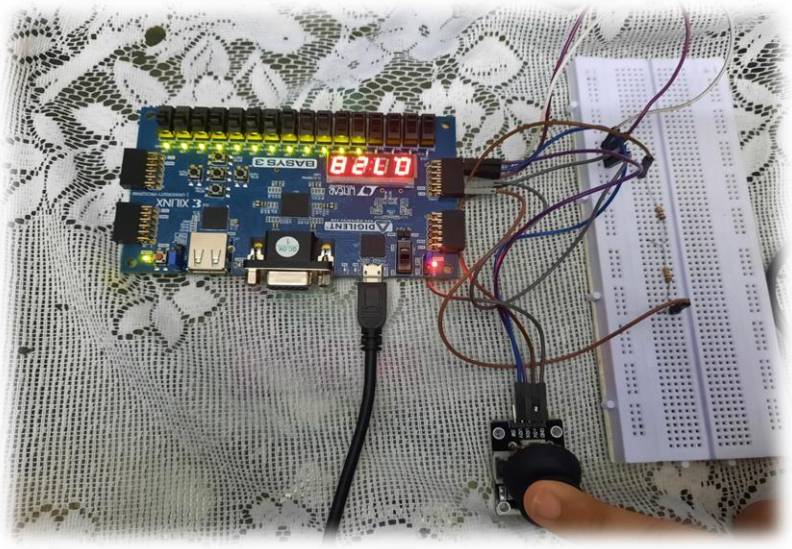

Figure 11: When joystick is moved in left direction

3.8. Push button as input:

When it comes to shooting fruit, we have found push button as our another input peripheral.

Push buttons are used to shoot arrows in order to hit the fruit.

3.9. SIMULATIONS:

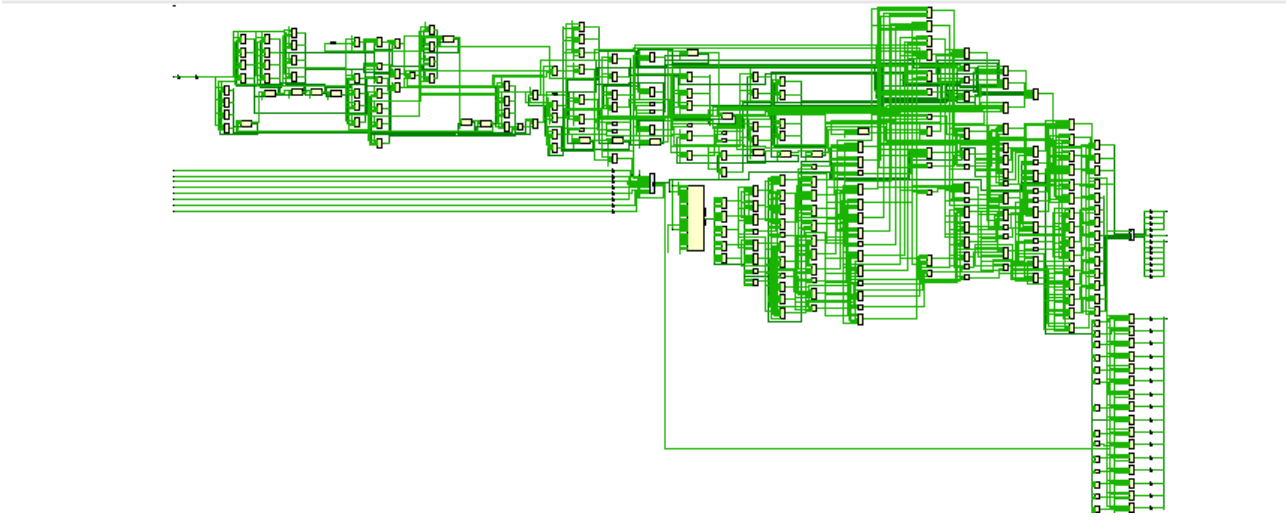

Figure 12: Schematic of input module

3.10. CODE SNIPPETS:

We used the code provided by our co batch mate for interfacing the joystick with the computer via FPGA basys 3 (GitHub link added in reference section). However, we have modified some parts of the code to meet our game requirements.

The modified code snippets are shown below.

1. The voltage is represented by data [15:12], so the case constraints that are implemented here are based on voltage. Since this voltage changes with joystick mobility, we confined the right and left joystick movements to specific voltage values.
2. Since the voltage produced in cases 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, and 15 is not synced with the voltage generated when the joystick moves left and right, we have set the value of the left and right buttons to zero.
3. We have attached snippets of only cases 1 and 2, which are similar to the remaining cases mentioned above.

```
114  always @( posedge(CLK100MHZ))
115  begin
116    if(ready == 1'b1)
117    begin
118      case (data[15:12])
119        1:  begin
120        LED <= 16'b11;
121        btnr<=1'b0;
122        btnl<=1'b0;
123        end
124        2: begin
125        LED <= 16'b111;
126        btnr<=1'b0;
127        btnl<=1'b0;
128        end
129        3:  begin
```

Figure 13: Case 1 and Case 2

1. Case 11 represent the voltage value when the joystick is moved to the left.

```
9:  LED <= 16'b1111111111;
10: begin
LED <= 16'b11111111111; //11 , left 1 right 0
btnr<=1'b0;
btnl<=1'b1;
end
```

Figure 14: Case 11

1. The default case represents the voltage generated whenever the joystick moves in the right direction, so we assign the value of the right button to 1 and the value of the left button to 0 in our code

```
default:
begin
 LED <= 16'b0;
 btnr <= 1'b1;
 btnl<= 1'b0;
 end
endcase
```

Figure 15: Default Case

3.11. INPUT BLOCK:

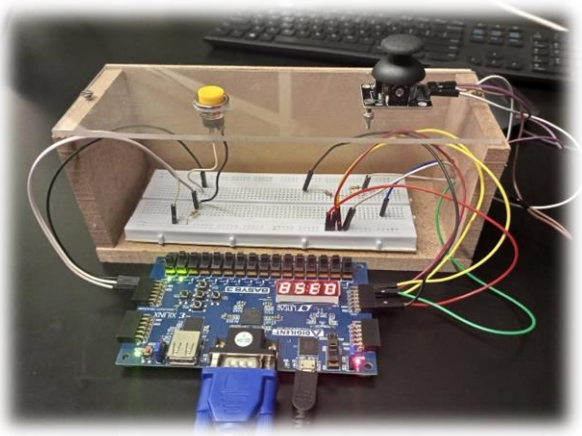

Figure 16: Input Block

SECTION 4: CONTROL BLOCK:

4.1. DESCRIPTION:

The control unit is the system's central hub, and is in charge of monitoring the game's flow. All of the game's finite state machines will be managed by the control block, which will regulate the start screen, game screen, and end screen. The FSM in our game is based on the functionality of a mealy machine, seeing as our lives next state is determined by his existing state and current input. Life is represented by a heart, and the number of lives deducted is controlled by our FSM.

4.2. STATES:

Finite state machines are being used to represent a system in which specific inputs cause specific changes in state. A state is a characterization of the status of a system that is expecting for a transition to be executed.

Lives exists in four states:

- 3 Hearts will be represented through: 00
- 2 hearts will be represented through: 01
- 1 heart will be represented through: 10
- When there is no heart, it will be represented through :11

4.3. STATE DIAGRAM:

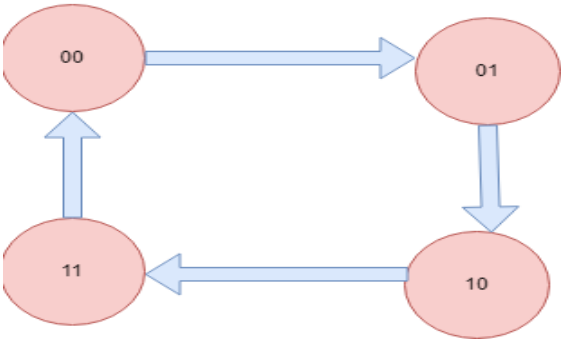

Figure 17: state diagram

- When the arrow collides with a obstacle or a barrier, one life is deducted, resulting in state change of heart.

4.4. STATE TABLE:

Input	Current State		Next State		Flip-flop Outputs	
I	A	B	A(t+1)	B(t+1)	T_A	TB
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	1	0	1	0	0	0
0	1	1	1	1	0	0
1	0	0	0	1	0	1
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	0	0	1	1

Figure 18: state table

4.5. STATE EQUATIONS:

1. T_A: IB
2. T_B: I

4.6. CIRCUIT DIAGRAM:

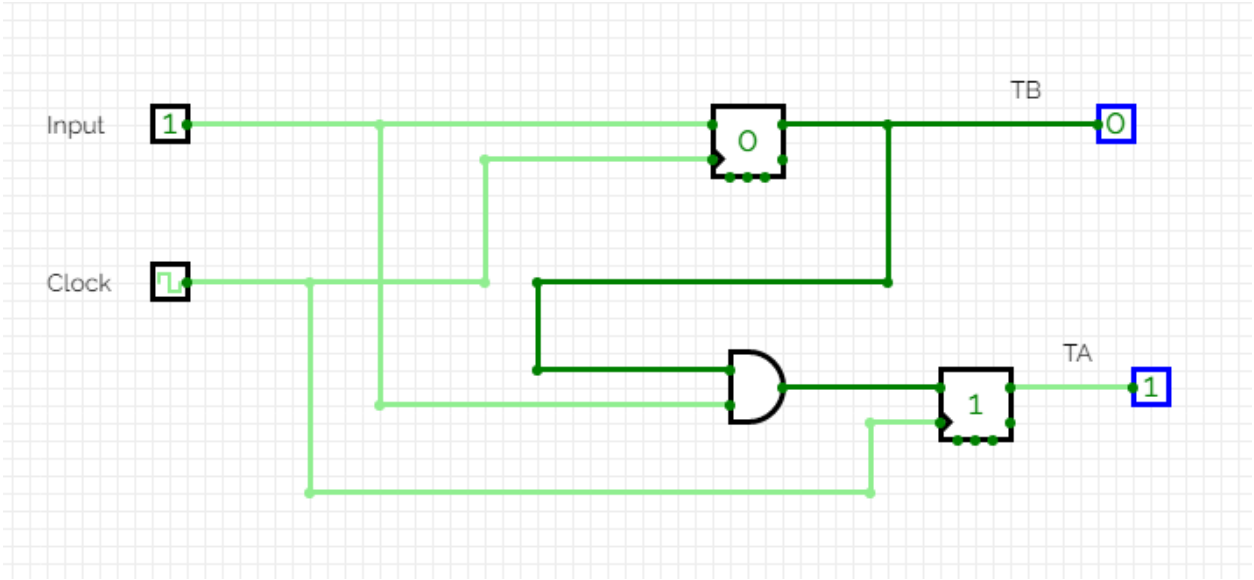

Figure 19: circuit diagram

SECTION 5: OUTPUT BLOCK:

5.1. OUTPUT:

Our game's output will be displayed on a computer screen (coded according to the screen size of 680x480), which will be connected to the FPGA via a VGA cable. In our game, we have three output screens. Our first screen displays the game's name, "fruit hunter." The screen is shifted to the main game screen by pressing the FPGA button "V17.". On the game's main screen, we began with generating the hunter, then modified our source code (included in the references), developed the logic and included fruits (Oranges, cherries and apples) in the game. Then, to complete our screen, we added obstacles. We added barriers between the obstacles to increase the complexity of our game.

5.2. PIXELS:

→In our game, we have three output screens

1. **FIRST SCREEN:**

Our first screen displays the name of our game, 'fruit hunter.' We used code from lab 11 to specify fruit hunter pixel by pixel. The output is displayed on a computer that is linked to the FPGA Basys 3 board via a VGA cable.

Figure 20: First screen

→Here the code snippet for our first screen:

```
wire F= ((y>=40 && y<=215) && (x>=18 && x<=58)) || ((y>=40 && y<=90) && (x>=18 &&
x<=124))|| ((y>=120 && y<=170) && (x>=18 && x<=108));
wire R= ((y>=40 && y<=215) && (x>=141 && x<=171)) || ((y>=40 && y<=80) && (x>=141
&& x<=247))||((y>=40 && y<=120) && (x>=217 && x<=247))||((y>=120 &&
y<=160)&&(x>=141 && x<=247))||((y>=160 && y<=215)&&(x>=197 && x<=227));
wire U= ((y>=40 && y<=215) && (x>=264 && x<=294)) || ((y>=170 && y<=215) &&
(x>=264 && x<=370))||((y>=40 && y<=215) && (x>=340 && x<=370));
wire I= ((y>=40 && y<=215) && (x>=387 && x<=443));
wire T= ((y>=40 && y<=100) && (x>=467 && x<=580)) || ((y>=100 && y<=215) &&
(x>=510 && x<=540));
wire H= ((y>=265 && y<=400) && (x>=13 && x<=43)) || ((y>=265 && y<=400) && (x>=74
&& x<=104)) || ((y>=300 && y<=350) && (x>=13 && x<=104));
wire U1= ((y>=265 && y<=400) && (x>=117 && x<=147)) || ((y>=360 && y<=400) &&
(x>=117 && x<=208)) || ((y>=265 && y<=400) && (x>=175 && x<=208));
wire N= ((y>=265 && y<=400) && (x>=221 && x<=241)) || ((y>=265 && y<=285) &&
(x>=221 && x<=255)) || ((y>=265 && y<=400)&&(x>=245 && x<=265)) ||((y>=370 &&
y<=400) && (x>=265 && x<=292))||((y>=265 && y<=400) && (x>=292 && x<=312));
wire T1= ((y>=265 && y<=325) && (x>=325 && x<=416)) || ((y>=325 && y<=400) &&
(x>=355 && x<=385));
wire E= ((y>=265 && y<=400) && (x>=429 && x<=459)) ||((y>=265 && y<=285) &&
(x>=429 && x<=526))||((y>=380 && y<=400) && (x>=429 && x<=526)) || ((y>=310 &&
y<=350) && (x>=429 && x<=526));
wire R1=((y>=265 && y<=400) && (x>=539 && x<=569)) || ((y>=265 && y<=295) &&
(x>=539 && x<=630))||((y>=265 && y<=320)&&(x>=600 && x<=630)) || ((y>=320 &&
y<=350) && (x>=539 && x<=630))||((y>=350 &&y<=400) && (x>=590 && x<=618));
 if (screen==0)
        begin
          RED<= F||R||U||I||T||H||U1||N||T1||E||R1 ;
          GREEN<=F||R||U||I||T;
          BLUE<=F||R||U||I||T;
```

2. **GAME SCREEN:**

On our game screen We required our game characters – (fruits, hunter, hearts, arrows and obstacles) to be in a particular format so that it could be used in Verilog. For that purpose, we used two softwares to convert our images to the hexadecimal (.mem) format.

First, we used GIMP to create a single XCF format image for all the characters present in our GUI. Each character was then cropped to the appropriate size/ pixels and converted to RAW image (.data) format using the same software. The RAW images created with the GIMP

software were then transformed into HEXADECIMAL format with the HxD software. This was done so that a (.mem) or, more simply, a hexadecimal file could be created that could be read by Verilog. This was done for each character: the hunter, the oranges, the cherries, and the apples. The (.mem) files were then accessed through the modules. For the background, we took help from the in-class lab we did regarding GUI – Lab-11. The usage of these programs and the creation of the GUI was done with the help of an internet resource we found (link is present in the references). However, the code, the files and the functionality was modified according to the need of our game. Our final code of the GUI can be accessed through the GitHub link provided in the references. Link to our GUI demonstration video could also be found in the references below.

→DESCRIPTION:

We created five different modules to generate our GUI. We created a display module for our background, in which we defined appropriate dimensions for our screen (took help from the lab 11 handout). We then created a module called "HunterSprite" to display our main character - the hunter. It reads the "huntermain.mem" file to define the pixels that will be used to display the hunter's image. By appropriately setting the coordinates on the screen, our hunter was displayed. For fruits, we defined a module called "FruitSprites," which reads three different files - "cherry.mem," "greenapple.mem," and "orangefinal.mem," to generate fruits on the screen. Similarly, we defined a module called "HiveSprite" for the obstacles, which reads the "hivetest.mem" file to generate the pixels that will be used to display the obstacles on the screen Besides that, we created a module for arrows called "arrowdisplay," and in this module, we implemented the collision and lives parts of our game. To display our background and all of the other modules on the screen at the same time, we wrote a module called "Top" that reads the "pal24bit.mem" file. The "pal24bit.mem" file is essentially a

palette file containing all of the colours (192 pixels) that will be used to display the background, hunter, fruits, and obstacles on the screen.

- Snippets of our screen's GUI:

Figure 21: Hunter

Figure 22: Fruits

Figure 23: obstacles

Figure 24: Main screen

3. **THIRD SCREEN:**

Our final screen reads "Game Over." We utilized code from lab 11 to specify it pixel by pixel. The output is displayed on a computer that is connected to the FPGA Basys 3 board via a VGA cable. When all of your lives are finished and the game is over, the main game screen will display a dialogue box that says "Gamend.".

Figure 25: “Game End” screen

5.3. BLOCK DIAGRAM:

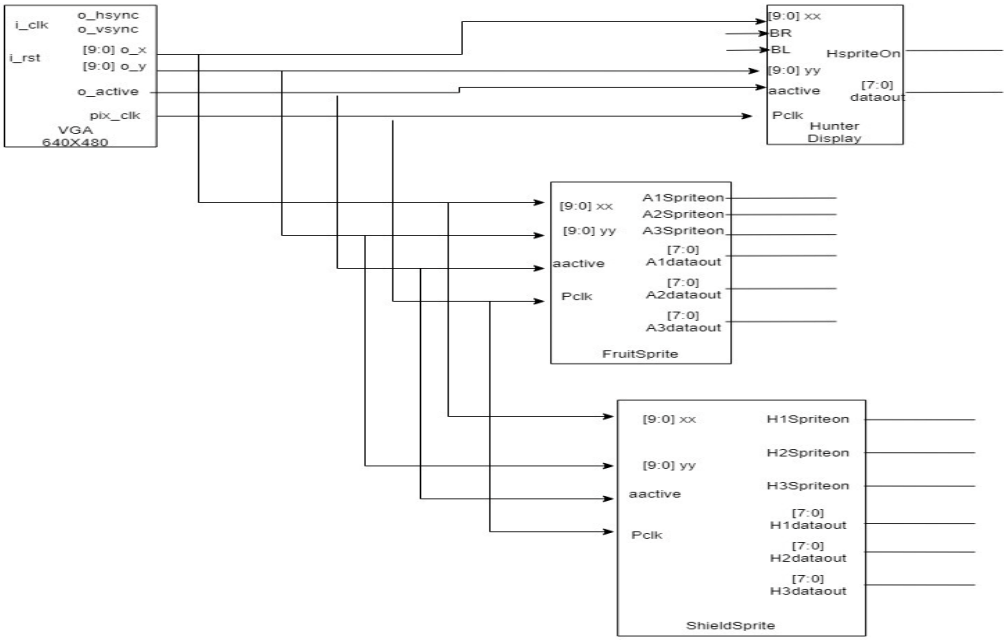

Figure 26: Top module Block diagram main screen .

5.4. SCHEMATIC:

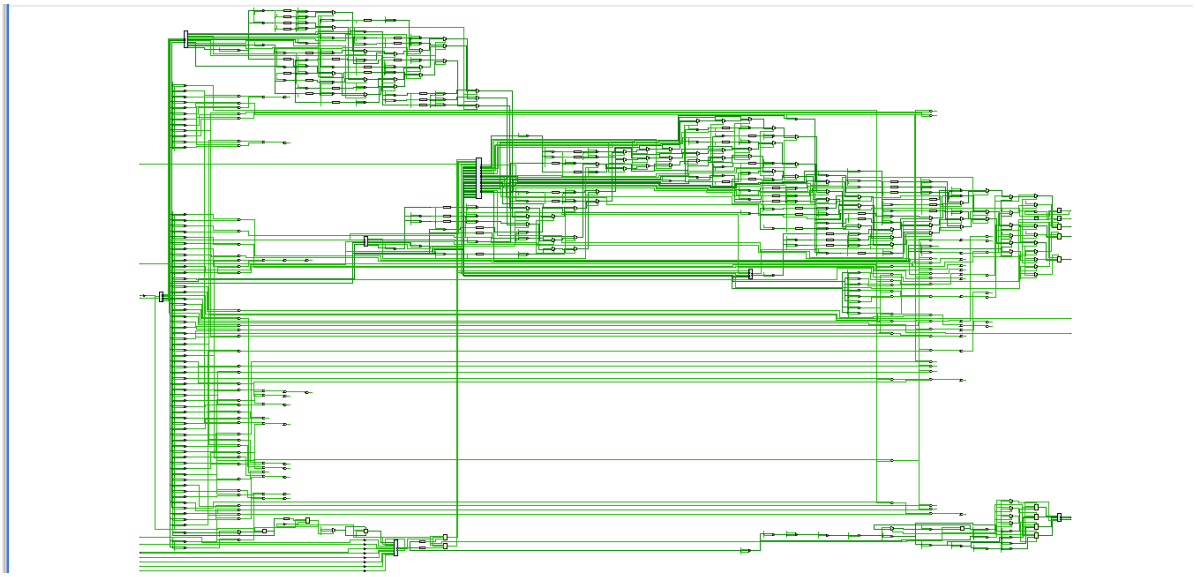

Figure 27: Schematic of GUI

SECTION 6: CONCLUSION:

Our game "Fruit Hunter" is inspired from the game space invaders, and bubble shooter that we all have played in our childhood. The main character of our game is a hunter who needs to collect all the fruits he can by overcoming the obstacles in the way. We will be controlling the hunter with the help of a 2-axis joystick. Since we only want the hunter to move in the x-axis, we can only move the joystick to the right, or left. Our game is an implementation of a mealy machine, as our character's movement - next state is determined by its current state, and the input provided by the joystick. In total, our hunter can be in one of the three states: idle, left or right, and these states will change as the joystick is moved. FSM is implemented in our game's lives section to depict the next state of our lives based on the current state, which is determined by whether the arrow hits fruits or any obstacles. Our output will be displayed on a (640x480) monitor that will be connected to the FPGA via a VGA cable. Since our GUI consisted of characters – hunter, fruits and obstacle, we created these characters on our own through different programs like Gimp and HxD where we converted

our characters into hexadecimal format which is recognized by Verilog. We have also created the required conditions to connect our joystick with the movement of our hunter on screen. We used a button to shoot the arrows, and whenever the arrow hit an obstacle or a barrier, one life was deducted. When all lives are depleted, our screen will display a game-ending dialogue box.

SECTION 7: ACKNOWLEDGEMENTS:

We would like to thank all of the DLD team's professors, especially Professor Farhan Khan and ma'am Hira Mustafa. We are grateful to our batchmate Muhammad Ali for his joystick code, which he developed under the supervision of professor Junaid Ahmed Memon. We would like to thank all of the RA'S of DLD who assisted us throughout the project.

SECTION 8: REFERENCE:

[1] "Demo video of our initial GUI – Google Drive Link"
https://drive.google.com/file/d/1bxfEt6lxzf_ZYejK16FeittNNHdF18jj/view?usp=share_link

[2] "Internet resource referenced for GUI creation"
https://github.com/AdrianFPGA/basys3

[3] "Joystick Module of our Co-Batch mate referenced for Input Module"
https://github.com/muhammadali74/Basys3-Joystick-Interfacing.git

[4] "Our Project's GitHub link"
https://github.com/batoolzehraaa/DLDPROJECT-22